CSC311 HW3

Xingkun Yin

1.

(a)

The dimension of $W^{(1)}$ is d as it is a matrix of $d * d$

The dimension of $W^{(2)}$ is d as it is a matrix of 1*d

The dimension of z_1 is 1 as it is a vector of $d * 1$

The dimension of z_2 is a vector of $d * 1$

(b)

There are a total of $d * d \ + \ d$ parameters

(c)

$\overline{y} \ = \ \frac{dL}{dy} = y - t$

$\overline{W}^{(2)} \ = \ \frac{dL}{dW^{(2)}} = \frac{dL}{dy} * \frac{dy}{dW^{(2)}} \ = \ \overline{y} * z_2$

$\overline{z_2} \ = \ \frac{dL}{dz_2} = \frac{dL}{dy} * \frac{dy}{dz_2} \ = \ \overline{y} * W^{(2)}$

$\overline{h} \ = \ \frac{dL}{dh} = \frac{dL}{dz_2} * \frac{dz_2}{dh} \ = \ \overline{z_2}$

$\overline{z_1} \ = \ \frac{dL}{dz_1} = \frac{dL}{dh} * \frac{dh}{dz_1} \ = \ \overline{z_2} * \sigma'(z_1)$

$\overline{W}^{(1)} \ = \ \frac{dL}{dW^{(1)}} = \frac{dL}{dz_1} * \frac{dz_1}{dW^{(1)}} \ = \ \overline{z_2} * \sigma'(z_1) * x^T$

$\overline{x} = \frac{dL}{dx} = \frac{dL}{dz_1} * \frac{dz_1}{dx} + \frac{dL}{dz_2} * \frac{dz_3}{dx} = \overline{z_2} * \sigma'(z_1) * W^{(1)} + \overline{y} * W^{(2)}$

2.

(a)

Solve $\frac{dy_k}{dz_{k'}}$

Given that $y_k = \frac{exp(z_k)}{\sum_{k'=1}^{K} exp(z_{k'})}$

$(1)\ If\ k' = k$

$$\frac{dy_k}{dz_{k'}} = \frac{exp(z_k) * \sum_{k'=1}^{K} exp(z_{k'}) - exp(z_k) * exp(z_{k'})}{(\sum_{k'=1}^{K} exp(z_{k'}))^2} = (\frac{exp(z_k)}{\sum_{k'=1}^{K} exp(z_{k'})}) * (1 - \frac{exp(z_k)}{\sum_{k'=1}^{K} exp(z_{k'})}) = y_k * (1 - y_k) = y_k - y_k^2$$

$(2)\ If\ k' \neq k$

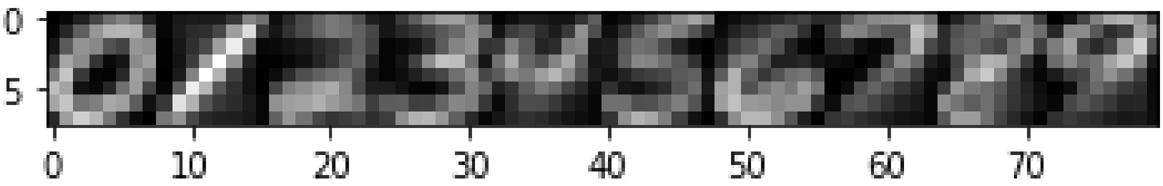
$$\frac{dy_k}{dz_{k'}} = \frac{0 * \sum_{k'=1}^{K} exp(z_{k'}) - exp(z_k) * exp(z_{k'})}{(\sum_{k'=1}^{K} exp(z_{k'}))^2} = (\frac{exp(z_k)}{\sum_{k'=1}^{K} exp(z_{k'})}) * (- \frac{exp(z_k')}{\sum_{k'=1}^{K} exp(z_{k'})}) = y_k * (-y_{k'}) = -y_k * y_{k'}$$

(b)

$$\frac{dL_{CE}}{dw_k} = \frac{dL_{CE}}{dy_k} * \frac{dy_k}{dz_k} * \frac{dz_k}{dw_k} = -\sum t_k \frac{1}{y_k} \frac{dy_k}{dz_k} * x$$

$$= ((-\sum_{K'=K} t_k * \frac{1}{y_k}) * (y_k - y_k^2) - \sum_{K' \neq K} \frac{t_{k'}}{y_k}(-y_k y_{k'})) * x = (-t_k + y_k \sum_{k'} t_{k'})x = (y_k - t_k)x$$

(3)
0.
Here are 10 plotted mean:



1.

1.

(a)
Test accuracy for k=1
0.96875
Train accuracy for k=1
1.0
(b)
Test accuracy for k=15
0.96
Train accuracy for k=15
0.9612857142857143

2.

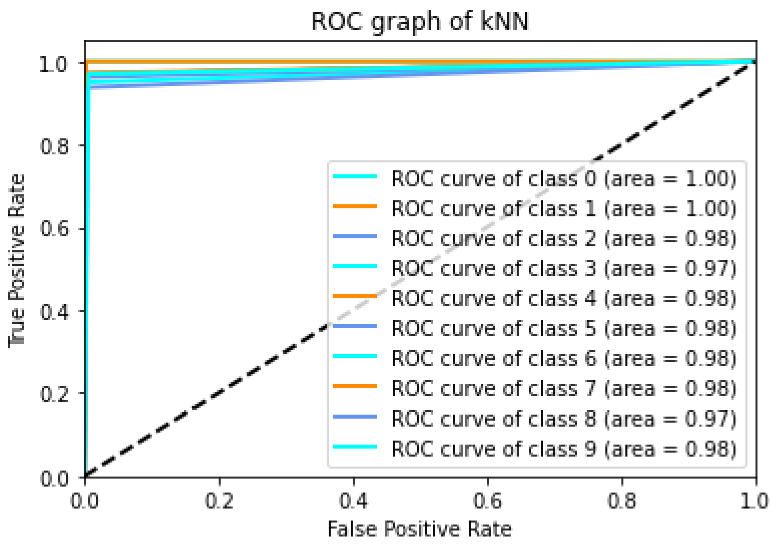
Randomly return one from all ties with the use of random.choice(). This is a fair method since all tied labels are equally likely to be the label of the new input. Therefore, I choose to use random in situation of a tie.

3.

k= 1
Test accuracy: 0.96875
Train accuracy: 1.0
10 fold average accuracy = 0.9658571428571427
k= 2
Test accuracy: 0.96325
Train accuracy: 0.9822857142857143
10 fold average accuracy = 0.9565714285714284
k= 3
Test accuracy: 0.968
Train accuracy: 0.983
10 fold average accuracy = 0.9628571428571429
k= 4
Test accuracy: 0.96725
Train accuracy: 0.9781428571428571
10 fold average accuracy = 0.9610000000000001
k= 5
Test accuracy: 0.96675
Train accuracy: 0.9778571428571429
10 fold average accuracy = 0.9621428571428572
k= 6

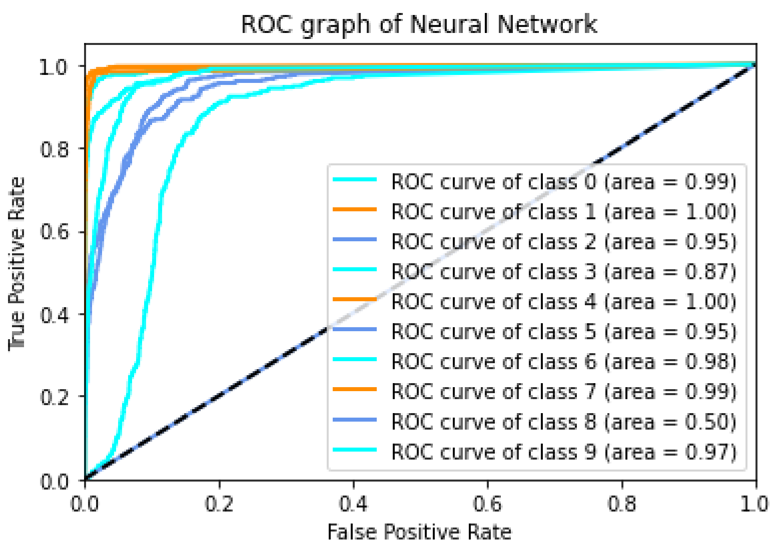
Test accuracy: 0.964
Train accuracy: 0.9728571428571429
10 fold average accuracy = 0.9602857142857143
k= 7
Test accuracy: 0.96375
Train accuracy: 0.9741428571428571
10 fold average accuracy = 0.9574285714285715
k= 8
Test accuracy: 0.963
Train accuracy: 0.9701428571428572
10 fold average accuracy = 0.9567142857142856
k= 9
Test accuracy: 0.9615
Train accuracy: 0.9692857142857143
10 fold average accuracy = 0.9559999999999998
k= 10
Test accuracy: 0.96175
Train accuracy: 0.9688571428571429
10 fold average accuracy = 0.9541428571428572
k= 11
Test accuracy: 0.9605
Train accuracy: 0.9658571428571429
10 fold average accuracy = 0.9537142857142855
k= 12
Test accuracy: 0.95925
Train accuracy: 0.9638571428571429
10 fold average accuracy = 0.9530000000000001
k= 13
Test accuracy: 0.9595
Train accuracy: 0.9635714285714285
10 fold average accuracy = 0.95
k= 14
Test accuracy: 0.95825
Train accuracy: 0.961
10 fold average accuracy = 0.948857142857143
k= 15
Test accuracy: 0.95925
Train accuracy: 0.9604285714285714
10 fold average accuracy = 0.9497142857142856

3.
kNN:
Accuracy: 0.97025




```
confusion matrix:
[[400   0   0   0   0   0   0   0   0   0]
 [  0 400   0   0   0   0   0   0   0   0]
 [  4   0 388   0   1   1   2   2   1   1]
 [  0   1   3 380   0   8   1   1   5   1]
 [  0   1   0   0 388   0   1   1   0   9]
 [  2   0   1   6   0 385   1   1   4   0]
 [  2   4   1   0   1   1 388   0   3   0]
 [  0   2   1   0   2   0   0 389   0   6]
 [  1   2   1   1   1  10   1   1 375   7]
 [  0   0   0   1   4   0   0   7   0 388]]
Precision:  0.9703887732518048
Recall:  0.9702500000000001
```

NN:

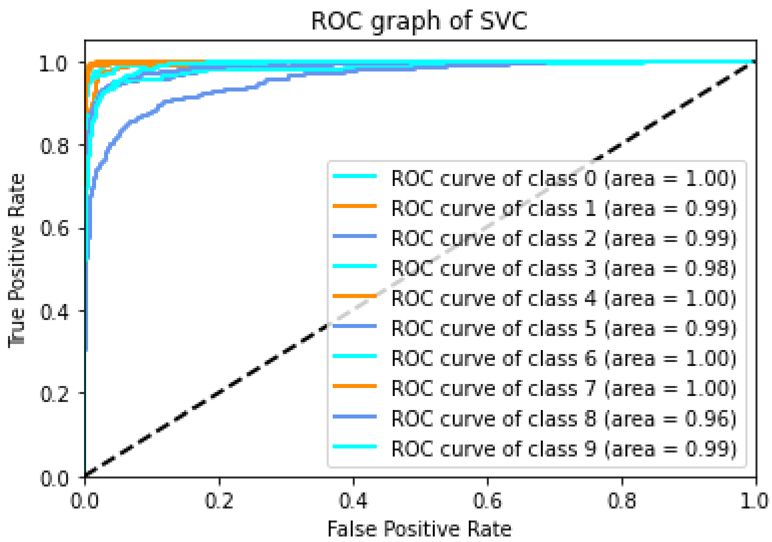


Train accuracy: 0.894
Test accuracy: 0.86375
confusion matrix:

```
[[390   4   4   0   0   0   2   0   0   0]
 [  0 395   0   0   1   0   1   0   0   3]
 [  1   1 377   6   0   3   6   2   0   4]
 [  2   0   8 366   0  15   0   2   0   7]
 [  0   1   1   0 387   0   3   0   0   8]
 [  3   0   1  10   0 383   0   0   0   3]
 [  3   5   2   0   3   2 385   0   0   0]
 [  0   0   1   0   3   0   0 390   0   6]
 [  0   9  13  14   2  10   0   2   0 350]
 [  0   2   0   2   4   2   0   8   0 382]]
```

Precision: 0.8096740440158637
Recall: 0.8637499999999999

SVM:



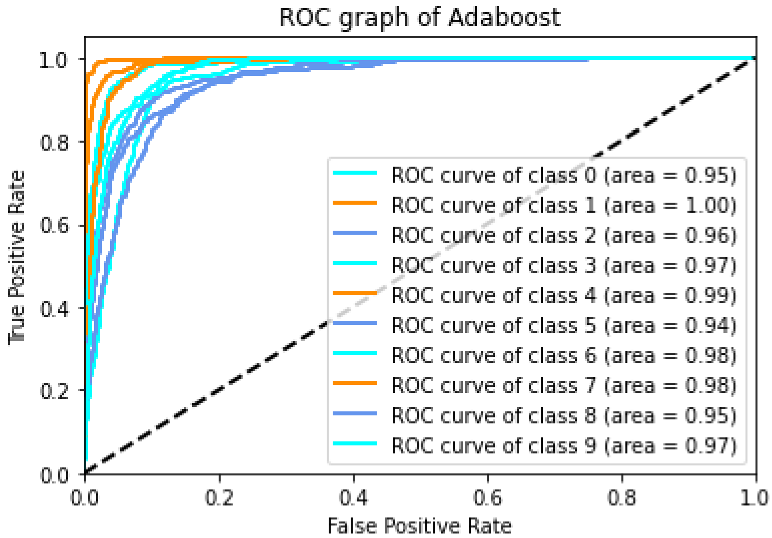
Accuracy: 0.93225
confusion matrix:

```
[[391   0   0   0   3   1   1   1   3   0]
 [  0 386   1   1   4   0   1   0   7   0]
 [  0   4 367   5   2   3   7   1   9   2]
 [  5   1  10 356   0  20   0   2   5   1]
 [  0   5   1   0 388   0   1   0   2   3]
 [  3   2   1  25   0 362   1   2   3   1]
 [  4   7   3   0   5   1 380   0   0   0]
 [  1   2   0   0   1   1   0 390   0   5]
 [  6   7   9  10   1  17   1   1 346   2]
 [  1   6   1   2   5   3   0   9  10 363]]
```

Precision: 0.9323913701166033

Recall: 0.93225

Adaboost:



Accuracy: 0.80975
confusion matrix:

```
[[362   1   1   1   0  25   3   0   7   0]
 [  2 376   0   2  12   0   3   1   4   0]
 [ 19   1 319   8   1  18  14   0  18   2]
 [  1   0  25 342   0  17   1   0  13   1]
 [  3   5   3   0 364   0   8   5   5   7]
 [  4   0   4  64   2 311   1   0  12   2]
 [146   5   6   0   4  39 198   0   2   0]
 [  0   1   1  18   3   1   0 281   5  90]
 [ 18   3   6  11   1  12   0   1 346   2]
 [  3   1   0   2  11   0   0  21  22 340]]
```

Precision: 0.8233458387457826
Recall: 0.80975

kNN performed the best in terms of accuracy and overall performance. Adaboost performed the worst.
It is a bit of a surprise but as expected. As we have limited samples and a small number of dimensions. Simpler methods like kNN are more suitable for this type of dataset while more complex models like neural network and adaboost require larger and more complex dataset to show its value to users.